FOCUS: A Constraint for Concentrating High Costs

Thierry Petit

TASC (Mines Nantes, LINA, CNRS, INRIA),
4, Rue Alfred Kastler, FR-44307 Nantes Cedex 3, France.
Thierry.Petit@mines-nantes.fr

Abstract. Many Constraint Programming models use integer cost variables aggregated in an objective criterion. In this context, some constraints involving exclusively cost variables are often imposed. Such constraints are complementary to the objective function. They characterize the solutions which are acceptable in practice. This paper deals with the case where the set of costs is a sequence, in which high values should be concentrated in a few number of areas. Representing such a property through a search heuristic may be complex and overall not precise enough. To solve this issue, we introduce a new constraint, FOCUS(X, y_c, len, k), where X is a sequence of n integer variables, y_c an integer variable, and len and k are two integers. To satisfy FOCUS, the minimum number of distinct sub-sequences of consecutive variables in X, of length at most len and that involve exclusively values strictly greater than k, should be less than or equal to y_c. We present two examples of problems involving FOCUS. We propose a complete filtering algorithm in $O(n)$ time complexity.

1 Introduction

Encoding optimization problems using Constraint Programming (CP) often requires to define cost variables, which are aggregated in an objective criterion. To be comparable, those variables have generally a totally ordered domain. They can be represented by integer variables. In this context, some constraints on cost variables are complementary to the objective function. They characterize the solutions which are acceptable in practice. For instance, to obtain balanced solutions several approaches have been proposed: Balancing constraints based on statistics [6, 11], as well as classical or dedicated cardinality constraints when the set of costs is a sequence [9, 8]. Some applications of these techniques are presented in [12, 7]. Representing such constraints, as well as solving efficiently the related problems, form an important issue because real-life problems are rarely "pure". In this context, CP is a well-suited technique. CP is generally robust to the addition of constraints, providing that they come up with filtering algorithms which impact significantly the search process.

Conversely to balancing constraints, in some problems involving a sequence of cost variables, the user wishes to minimize the number of sub-sequences of consecutive variables where high cost values occur.

Example 1. We consider a problem where some activities have to be scheduled. Each activity consumes an amount of resource. The total amount of consumption at a given time is limited by the capacity of the machine that produces the resource. If the time

window where activities have to be scheduled is fixed, in some cases not all the activities can be scheduled, because there is not enough quantity of resource to perform all the activities on time. Assume that, in this case, we rent a second machine to solve the problem. In practice, it is often less costly to rent such a machine within a package, that is, during consecutive periods of time. If you rent the machine during three consecutive days, the price will be lower than the price of three rentals of one day in three different weeks. Moreover, such packages are generally limited, e.g., the maximum duration of one rental is one week. If you exceed one week then you need to sign two separate contracts. Thus, to satisfy the end-user, a solution should both limit and concentrate the exceeds of resource consumption, given a maximum rental duration. ⊛

In Example 1, a solution minimizing the exceeds with many short and disjoint rental periods will be more expensive for the end-user than a non-minimum solution where rentals are focused on a small number of periods. Such a constraint cannot be easily simulated with a search strategy, a fortiori when the duration of packages is limited. Furthermore, to solve instances, search heuristics are generally guided by the underlying problem (in Example 1, the cumulative problem). Our contribution is a generic, simple and effective way to solve this issue. It comes in two parts.

1. A new constraint, FOCUS(X, y_c, len, k), where X is a sequence of integer variables, y_c an integer variable, and len and k two integer values. y_c limits the number of distinct sub-sequences in X, each of length at most len, involving exclusively values strictly greater than k. More precisely, the minimum possible number of such sub-sequences should be less than or equal to y_c, while any variable in X taking a value $v > k$ belongs to exactly one sub-sequence.
2. A $O(n)$ Generalized Arc-Consistency (GAC) filtering algorithm for FOCUS.

Section 2 defines the FOCUS constraint. Section 3 presents two examples of use. In section 4, we present the $O(n)$ complete filtering algorithm for FOCUS. Section 5 introduces some variations of FOCUS, namely the case where len is a variable and the case where the constraint on the variable y_c is more restrictive. We discuss the related work and propose an automaton-based reformulation of FOCUS. Our experiments, in Section 7, show the importance of providing FOCUS with a complete filtering algorithm.

2 The FOCUS constraint

Given a sequence of integer variables $X = \langle x_0, x_1, \ldots, x_{n-1} \rangle$ of length $|X| = n$, an instantiation of X is a valid assignment, *i.e.*, a sequence of values $I[X] = \langle v_0, v_1, \ldots v_{n-1} \rangle$ such that $\forall j \in \{0, 1, \ldots, n-1\}, v_j$ belongs to $D(x_j)$, the domain of x_j.

Definition 1 (FOCUS)**.** *Given $X = \langle x_0, x_1, \ldots, x_{n-1} \rangle$, let y_c be an integer variable such that $0 \leq y_c \leq |X|$, len be an integer such that $1 \leq len \leq |X|$, and $k \geq 0$ be an integer. Given an instantiation $I[X] = \langle v_0, v_1, \ldots v_{n-1} \rangle$, and a value v_c assigned to y_c,* FOCUS*$(I[X], v_c, len, k)$ is satisfied if and only if there exists a set S of* **disjoint** *sequences of consecutive variables in X such that three conditions are all satisfied:*

1. *Number of sequences: $|S| \leq v_c$*

2. *One to one mapping of all values strictly greater than k:*
   $\forall j \in \{0, 1, \ldots, n-1\}$, $v_j > k \Leftrightarrow \exists s_i \in S$ *such that* $x_j \in s_i$
3. *Length of a sequence in S:* $\forall s_i \in S$, $1 \leq |s_i| \leq len$.

If $len = |X|$, y_c limits the number of disjoint maximum length sequences where all the variables are assigned with a value strictly greater than k. Otherwise, len limits the length of the sequences counted by y_c. Example 2 illustrates the two cases.

Example 2. Let $I[X] = \langle 1, 3, 1, 0, 1, 0 \rangle$. FOCUS($I[X], \langle 2 \rangle, 6, 0$) is satisfied since we can have 2 disjoint sequences of length ≤ 6 of consecutive variables with a value > 0, *i.e.*, $\langle x0, x1, x2 \rangle$, and $\langle x4 \rangle$. FOCUS($I[X], \langle 2 \rangle, 2, 0$) is violated since it is not possible to include all the strictly positive variables in X with only 2 sequences of length ≤ 2. $\circledast$

3 Examples of Use

Constraints and Music An important field in musical problems is automatic composition and harmonization. In many cases, the end-user wishes to obtain the maximum length sequences of measures where her rules are minimally violated. We consider the example of the *sorting chords* problem [5, 13]. The goal is to sort n distinct chords. A chord is a set of at most p notes played simultaneously. p can vary from one chord to another. The sort should reduce as much as possible the number of notes changing between two consecutive chords. The musician may be particularly interested by large sub-sequences of consecutive chords where there is at most $nchange$ different notes between two consecutive chords, and thus she aims at concentrating high changes in a few number of areas. We represent the sequence by n variables $Chords = \langle ch_0, ch_1, \ldots, ch_{n-1} \rangle$, such as each variable can be instantiated with any of the chords. The constraint ALLDIFF($Chords$) [10] imposes that all ch_i's are pairwise distinct. $nchange$ is at least 1. Therefore, we define the cost between two consecutive chords in the sequence as the number of changed notes less one. It is possible to compute that cost for each pair of chords (the number of costs is $n \times (n-1)/2$), and link this value with the chords through a ternary table constraint. We call such a constraint COSTC$_i(ch_i, ch_{i+1}, cost_i)$, where $cost_i \in X$ is the integer variable representing the cost of the pair (ch_i, ch_{i+1}). Its domain is the set of distinct cost values considered when COSTC$_i(ch_i, ch_{i+1}, cost_i)$ is generated. FOCUS is imposed on $X = \langle cost_0, cost_1, \ldots, cost_{n-2} \rangle$, in order to concentrate high costs (for instance costs > 2, that is $nchange = 3$) in a few number of areas. If their length is not constrained $len = |X|$, otherwise the end-user can fix a smaller value. The constraint model is:
ALLDIFF($Chords$) $\wedge$ $\forall i \in \{0, 1, \ldots, n-2\}$ COSTC$_i(ch_i, ch_{i+1}, cost_i)$
$\wedge$ FOCUS($X, y_c, len, 2$) $\wedge$ $sum = \sum_{i \in \{0,1,\ldots,n-2\}} cost_i$
Two objectives can be defined: $minimize(sum)$ and $minimize(y_c)$.

Over-Loaded Cumulative Scheduling In Example 1 of the Introduction, the core of the problem can be represented using the SOFTCUMULATIVE constraint [2]. The time window starts at time 0 and ends at a given strictly positive integer, the $horizon$ (*e.g.*, 160 points in times which are, for instance, the total amount of hours of 4 weeks of

work). Activities $a_k \in A$ are represented by three variables: starting time, duration, resource consumption. Using SOFTCUMULATIVE, some intervals of time $I_i \in \mathcal{I}$ (*e.g.*, one day of 8 hours), one to one mapped with cost variables $cost_i \in X$, are given by the user. A cost measures how much the capacity $capa$ is exceeded within the interval I_i. The maximum value in the domain of each variable $cost_i$ expresses the maximum allowed excess. In [2], several definitions of costs are proposed. We can for instance define $cost_i$ as the exceed of the maximum over-loaded hour in the interval I_i.

The constraint related to the additional machine is FOCUS($X, y_c, len, 0$), where $X = \langle cost_0, cost_1, \ldots, cost_{|I|-1} \rangle$. len is the maximum duration of one rental, e.g., 5 days, that is, $len = 40$ (if the time unit is one hour). The constraint model is:
SOFTCUMULATIVE($A, X, \mathcal{I}, horizon$) $\wedge$ FOCUS($X, y_c, len, 0$)
$\wedge\ sum = \sum_{i \in \{0,1,\ldots,n-2\}} cost_i$

Two objectives can be defined: $minimize(sum)$ and $minimize(y_c)$.

4 Linear Filtering Algorithm

4.1 Characterization of Sequences

Notation 1 (Status of a variable) *Let $X = \langle x_0, x_1, \ldots, x_{n-1} \rangle$ be a sequence of integer variables and k an integer. According to k, a variable $x_i \in X$ is:* Penalizing *(P_k) if and only if the minimum value in its domain* $\min(x_i)$ *is such that* $min(x_i) > k$. Neutral *(N_k) if and only if the maximum value in its domain* $\max(x_i)$ *is such that* $max(x_i) \leq k$. Undetermined *(U_k) if* $min(x_i) \leq k$ *and* $max(x_i) > k$.

Definition 2 (Maximum σ-sequence). *Let $X = \langle x_0, x_1, \ldots, x_{n-1} \rangle$ be a sequence of integer variables, k an integer, and $\sigma \subseteq \{\mathrm{P}_k, \mathrm{N}_k, \mathrm{U}_k\}$. A* σ-sequence $\langle x_i, x_{i+1}, \ldots, x_j \rangle$ *of X is a sequence of consecutive variables in X such that all variables have a status in σ and for all status $s \in \sigma$ there exists at least one variable in the sequence having the status s. It is* maximum *if and only if the two following conditions are satisfied:*

1. *If $i > 0$ then the status of x_{i-1} is not in σ.*
2. *If $j < n - 1$ then the status of x_{j+1} is not in σ.*

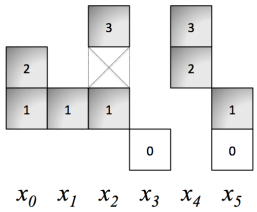

Fig. 1. $X = \langle x_0, x_1 \ldots, x_5 \rangle$ is a maximum-length $\{\mathrm{N}_0, \mathrm{P}_0, \mathrm{U}_0\}$-sequence, which contains one maximum-length $\{\mathrm{N}_0, \mathrm{P}_0\}$-sequence $\langle x_0, x_1, \ldots, x_4 \rangle$, two maximum-length $\{\mathrm{P}_0\}$-sequences $\langle x_0, x_1, x_2 \rangle$ and $\langle x_4 \rangle$, one maximum length $\{\mathrm{P}_0, \mathrm{U}_0\}$-sequence $\langle x_4, x_5 \rangle$, one maximum-length $\{\mathrm{N}_0\}$-sequence $\langle x_3 \rangle$ and one maximum-length $\{\mathrm{U}_0\}$-sequence $\langle x_5 \rangle$.

Figure 1 illustrates Definition 2. The picture shows the domains in a sequence of $n = 6$ variables, with $k = 0$. Grey squares are values strictly greater than k, while the white ones correspond to values less than or equal to k.

Definition 3 (Focus cardinality of a σ-sequence). *Given a sequence of variables X and len and k two integer values, the focus cardinality $card(X, len, k)$ is the minimum value v_c such that* FOCUS*(X, v_c, len, k) has a solution.*

We can evaluate the focus cardinality according to the different classes of sequences.

Property 1. Given a $\{\mathrm{P}_k\}$-sequence Y, $card(Y, len, k) = \lceil \frac{|Y|}{len} \rceil$.

Proof. $\lfloor \frac{|Y|}{len} \rfloor$ is the minimum number of distinct sequences of consecutive variables of length len within Y, and the remainder r of $\frac{|Y|}{len}$ is such that $0 \leq r < len$. □

Notation 2 *Given a $\{\mathrm{N}_k,\mathrm{P}_k\}$-sequence X, $P_k(X)$ denotes the set of disjoint maximum $\{\mathrm{P}_k\}$-sequences extracted from X.*

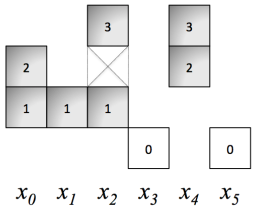

Fig. 2. A $\{\mathrm{N}_0,\mathrm{P}_0\}$-sequence $X = \langle x_0, x_1, \ldots, x_5 \rangle$. $card(X, 1, 0) = \sum_{Y \in P_0(X)} \lceil \frac{|Y|}{1} \rceil = 3 + 1 = 4$. $card(X, 2, 0) = \sum_{Y \in P_0(X)} \lceil \frac{|Y|}{2} \rceil = 2+1 = 3$. $card(X, 4, 0) = \sum_{Y \in P_0(X)} \lceil \frac{|Y|}{4} \rceil = 2$.

Property 2. Given a $\{\mathrm{N}_k,\mathrm{P}_k\}$-sequence X, $card(X, len, k) = \sum_{Y \in P_k(X)} card(Y, len, k)$.

Proof. By definition of a $\{\mathrm{N}_k,\mathrm{P}_k\}$-sequence, variables outside these sequences take a value less than or equal to k. From Property 1, the property holds. □

Figure 2 illustrates Property 2. When X is a $\{\mathrm{N}_k,\mathrm{P}_k\}$-sequence, for instance an instantiation $I[X]$, and y_c is fixed to a value v_c, we can encode a checker for FOCUS, based on the computation of the focus cardinality of X.[1]

The correctness of Algorithm 1 is proved by Properties 1 and 2. Its time complexity is obviously $O(n)$. The computation for of a $\{\mathrm{N}_k,\mathrm{P}_k,\mathrm{U}_k\}$-sequence requires to prove some properties. In Figure 1, we have $len = 1$. Depending whether x_5 is assigned to 0 or to 1, the value of y_c satisfying FOCUS$(X, y_c, len, 0)$ is either 4 or 5.

4.2 Feasibilty and Filtering Algorithm

Definition 4. *Given $x_i \in X$, $i \in \{0, 1, \ldots, n-1\}$, and $v \in D(x_i)$,*

[1] For a end-user, we can provide a set of sub-sequences corresponding to the focus cardinality: the algorithm is similar to Algorithm 1 (we store the sequences instead of counting them).

Algorithm 1: isSatisfied({N$_k$,P$_k$}-sequence $X = \langle x_0, x_1, \ldots, x_{n-1}\rangle$, v_c, len, k): boolean

```
1  Integer nb := 0;
2  Integer size := 0;
3  Boolean prevpk := false;
4  for Integer i := 0; i < n; i := i + 1 do
5      if min(x_i) > k then
6          size := size + 1;
7          prevpk := true;
8      else
9          if prevpk then nb := nb + ⌈size/len⌉;
10         size := 0;
11         prevpk := false;
12 if prevpk then nb := nb + ⌈size/len⌉;
13 return nb ≤ v_c; // focus cardinality of X
```

- $\underline{p}(x_i, v)$ *is the focus cardinality* $card(\langle x_0, x_1, \ldots x_i\rangle, len, k)$ *of the prefix sequence* $\langle x_0, x_1, \ldots x_i\rangle$ *when* $x_i = v$.
- $\underline{s}(x_i, v)$ *is the focus cardinality* $card(\langle x_i, x_{i+1}, \ldots x_{n-1}\rangle, len, k)$ *of the suffix sequence* $\langle x_i, x_{i+1}, \ldots x_{n-1}\rangle$ *when* $x_i = v$.

The remaining of this section is organized as follows. First, we show how we can check the feasibility of FOCUS and enforce a complete filtering of domains of variables in X and $D(y_c)$, provided we have the data of Definition 4. Then, we explain how such a data and the filtering algorithm can be obtained in $O(n)$.

Given $x_i \in X$, the two quantities of Definition 4 can have, each, at most two distinct values: one for the values in $D(x_i)$ strictly greater than k, one for the values in $D(x_i)$ less than or equal to k. This property holds by the definition of the constraint FOCUS itself (Definition 1): From the point of view of FOCUS, value $k+1$ or value $k+1000$ for x_i are equivalent. We use a new notation, which groups values of Definition 4.

Notation 3 *Given* $x_i \in X$,

- $\underline{p}(x_i, v_>)$ *is the value of* $\underline{p}(x_i, v)$ *for all* $v \in D(x_i)$ *such that* $v > k$, *equal to* $n+1$ *if there is no value* $v > k$ *in* $D(x_i)$.
- $\underline{p}(x_i, v_\leq)$ *is the value of* $\underline{p}(x_i, v)$ *for all* $v \in D(x_i)$ *such that* $v \leq k$, *equal to* $n+1$ *if there is no value* $v \leq k$ *in* $D(x_i)$.

Similarly, we use the notations $\underline{s}(x_i, v_>)$ *and* $\underline{s}(x_i, v_\leq)$ *for suffix sequences.*

Given such quantities for the last variable (or the first if we consider suffixes), we obtain a feasibility check for FOCUS. Their computation is explained in next section.

Algorithm 2: isSatisfied($X = \langle x_0, x_1, \ldots, x_{n-1}\rangle$, y_c, len, k): boolean

```
1  return min(p(x_{n-1}, v_>), p(x_{n-1}, v_≤)) ≤ max(y_c) ;
```

We use the following notation: $minCard(X) = \min(\underline{p}(x_{n-1}, v_>), \underline{p}(x_{n-1}, v_\leq))$. With that data, we can update $\min(y_c)$ to $\min(minCard(X), \min(y_c))$. Then from

Definition 1, all the values in $D(y_c)$ have a valid support on FOCUS (by definition any value of y_c greater than $minCard(X)$ satisfies the constraint). By applying $O(n)$ times Algorithm 2, in order to study each variable x_i in X successively restricted to the range of values $\leq k$ as well as the range of values $> k$, we perform a complete filtering.

Lemma 1. . *Given a* U_k *variable* x_i*, let* $X_i^{>} = \{x_0^{>}, x_1^{>}, \ldots, x_{n-1}^{>}\}$ *be the set of variables derived from* X *such that* $\forall j \in \{0, 1, \ldots, i-1, i+1, \ldots, n-1\}$*,* $D(x_j^{>}) = D(x_j)$ *and* $D(x_i^{>}) = D(x_i) \cap [k+1, \max(x_i)]$*. If* $minCard(X_i^{>}) > \max(y_c)$ *then the range* $[k+1, \max(x_i)]$ *can be removed from* $D(x_i)$*.*

Lemma 2. *Given a* U_k *variable* x_i*, let* $X_i^{>} = \{x_0^{>}, x_1^{>}, \ldots, x_{n-1}^{>}\}$ *be the set of variables derived from* X *such that* $\forall j \in \{0, 1, \ldots, i-1, i+1, \ldots, n-1\}$*,* $D(x_j^{\leq}) = D(x_j)$ *and* $D(x_i^{\leq}) = D(x_i) \cap [\min(x_i), k]$*. If* $minCard(X_i^{\leq}) > \max(y_c)$ *then the range* $[\min(x_i), k]$ *can be removed from* $D(x_i)$*.*

Proof (Lemmas 1 and 2). Direct consequence of Definitions 1 and 3. □

Given $O(\Phi)$ the time complexity of an algorithm computing $minCard(X)$, we can perform the complete filtering of variables in $X \cup \{y_c\}$ in $O(n \times \Phi)$, where $n = |X|$. We now show how to decrease the whole time complexity to $O(n)$, both for computing $minCard(X)$ and shrink the domains of all the variables in X. Given $x_i \in X$, the first idea is to compute $\underline{p}(x_i, v_>)$ from $\underline{p}(x_{i-1}, v_>)$ and $\underline{p}(x_{i-1}, v_\leq)$. To do so, we have to estimate the minimum length of a $\{\mathrm{P}_k\}$-sequence containing x_i, within an instantiation of $\langle x_0, x_1, \ldots, x_i \rangle$ of focus cardinality $\underline{p}(x_i, v_>)$. We call this quantity $\underline{plen}(x_i)$. Next lemmas provide the values of $\underline{p}(x_i, v_>)$, $\underline{p}(x_{i-1}, v_\leq)$ and $\underline{plen}(x_i)$, from x_{i-1}.

Lemma 3 (case of x_0).

- *If* x_i *is a* $\{\mathrm{P}_k\}$*-variable,* $\underline{p}(x_0, v_\leq) = n+1$*,* $\underline{p}(x_0, v_>) = 1$ *and* $\underline{plen}(x_0) = 1$*.*
- *If* x_i *is a* $\{\mathrm{N}_k\}$*-variable,* $\underline{p}(x_0, v_\leq) = 0$*,* $\underline{p}(x_0, v_>) = n+1$ *and* $\underline{plen}(x_0) = 0$*.*
- *If* x_i *is a* $\{\mathrm{U}_k\}$*-variable,* $\underline{p}(x_0, v_\leq) = 0$*,* $\underline{p}(x_0, v_>) = 1$ *and* $\underline{plen}(x_0) = 1$*.*

Proof. If x_0 takes a value $v > k$ then by Definition 4 $\underline{p}(x_0, v_>) = 1$ and $\underline{plen}(x_0) = 1$. Otherwise, there is no $\{\mathrm{P}_k\}$-sequence containing x_0 and $\underline{plen}(x_0) = 0$: We use the convention $\underline{p}(x_0, v_>) = n+1$ (an absurd value: the max. number of sequences in X is n). If x_0 belongs to a $\{\mathrm{P}_k\}$-sequence then $\underline{p}(x_0, v_\leq) = n+1$. □

Lemma 4 (computation of $\underline{p}(x_i, v_\leq)$, $0 < i < n$). *If* x_i *is a* $\{\mathrm{P}_k\}$*-variable then* $\underline{p}(x_i, v_\leq) = n+1$*. Otherwise,* $\underline{p}(x_i, v_\leq) = \min(\underline{p}(x_{i-1}, v_>), \underline{p}(x_{i-1}, v_\leq))$*.*

Proof. If x_i belongs to a $\{\mathrm{P}_k\}$-sequence then x_i does not take a value $v \leq k$, thus $\underline{p}(x_0, v_\leq) = n+1$. If there exists some values less than or equal to k in $D(x_i)$, assigning one such value to x_i leads to a number of $\{\mathrm{P}_k\}$-sequences within the prefix sequence $\langle x_0, x_1, \ldots, x_i \rangle$ which does not increase compared with the sequence $\langle x_0, x_1, \ldots, x_{i-1} \rangle$. Thus, $\underline{p}(x_i, v_\leq) = \min(\underline{p}(x_{i-1}, v_>), \underline{p}(x_{i-1}, v_\leq))$. □

Lemma 5 (computation of $\underline{p}(x_i, v_>)$ and $\underline{plen}(x_i)$, $0 < i < n$). *We have:*

- If x_i is a $\{\mathrm{N}_k\}$-variable then $\underline{p}(x_i, v_>) = n+1$ and $\underline{plen}(x_i) = 0$.

– Otherwise,
  - If $\underline{plen}(x_{i-1}) = len \vee \underline{plen}(x_{i-1}) = 0$ then $\underline{p}(x_i, v_>) = \min(\underline{p}(x_{i-1}, v_>) + 1, \underline{p}(x_{i-1}, v_\leq) + 1)$ and $\underline{plen}(x_i) = 1$.
  - Otherwise $\underline{p}(x_i, v_>) = \min(\underline{p}(x_{i-1}, v_>), \underline{p}(x_{i-1}, v_\leq) + 1)$ and:
    * If $\underline{p}(x_{i-1}, v_>) \leq \underline{p}(x_{i-1}, v_\leq)$ then $\underline{plen}(x_i) = \underline{plen}(x_{i-1}) + 1$.
    * Else $\underline{plen}(x_i) = 1$.

Proof. If x_i is a $\{\mathrm{N}_k\}$-variable then it cannot take a value $> k$. By convention $\underline{p}(x_i, v_>) = n + 1$ and $\underline{plen}(x_i) = 0$. Otherwise, recall that from Definition 3, the focus cardinality is the minimum possible number of $\{\mathrm{P}_k\}$-sequences. If $0 < \underline{plen}(x_{i-1}) < len$ the last $\{\mathrm{P}_k\}$-sequence can be extended by variable x_i within an assignment having the same focus cardinality than the one of $\langle x_0, x_1, \ldots, x_{i-1}\rangle$, thus $\min(\underline{p}(x_{i-1}, v_>), \underline{p}(x_{i-1}, v_\leq) + 1)$ and $\underline{plen}(x_i)$ is updated so as to remain the minimum length of a $\{\mathrm{P}_k\}$-sequence containing x_i in an instantiation of $\langle x_0, x_1, \ldots, x_i\rangle$ of focus cardinality $\underline{p}(x_i, v_>)$. Otherwise, the focus cardinality will be increased by one if x_i takes a value $v > k$. We have $\underline{p}(x_i, v_>) = \min(\underline{p}(x_{i-1}, v_>) + 1, \underline{p}(x_{i-1}, v_\leq) + 1)$. Since we have to count a new $\{\mathrm{P}_k\}$-sequence starting at x_i, $\underline{plen}(x_i) = 1$. □

Given $X = \langle x_0, x_1, \ldots, x_{n-1}\rangle$, Algorithm 3 uses the Lemmas to computes in $O(n)$ the quantities. It returns a matrix $cards$ of size $n \times 3$, such that at each index i:

$$cards[i][0] = \underline{p}(x_i, v_\leq);\ cards[i][1] = \underline{p}(x_i, v_>);\ cards[i][2] = \underline{plen}(x_i)$$

We can then compute for each x_i $\underline{s}(x_i, v_>)$, $\underline{s}(x_{i-1}, v_\leq)$ and $\underline{slen}(x_i)$ (the equivalent of $\underline{plen}(x_i)$ for suffixes), by using Lemmas 3, 4 and 5 with X sorted in the reverse order: $\langle x_{n-1}, x_{n-2}, \ldots, x_0\rangle$. To estimate for each variable $minCard(X_i^{\leq})$ and $minCard(X_i^{>})$, we have to aggregate the quantities on prefixes and suffixes.

Property 3. $minCard(X_i^{\leq}) = \underline{p}(x_i, v_\leq) + \underline{s}(x_i, v_\leq)$ and $minCard(X_i^{>})$ is equal to:

– $\underline{p}(x_i, v_>) + \underline{s}(x_i, v_>) - 1$ if and only if $\underline{plen}(x_i) + \underline{slen}(x_i) - 1 \leq len$.
– $\underline{p}(x_i, v_>) + \underline{s}(x_i, v_>)$ otherwise.

Proof. Any pair of instantiations of focus cardinality respectively equal to $\underline{p}(x_i, v_\leq)$ and $\underline{s}(x_i, v_\leq)$ correspond to disjoint $\{\mathrm{P}_k\}$-sequences (which do not contain x_i). Thus the quantities are independent and can be summed. With respect to $minCard(X_i^{>})$, the last current $\{\mathrm{P}_k\}$-sequence taken into account in $\underline{p}(x_i, v_>)$ and $\underline{s}(x_i, v_>)$ contains x_i. Thus, their union (of length $\underline{plen}(x_i) + \underline{slen}(x_i) - 1$) forms a unique $\{\mathrm{P}_k\}$-sequence, from which the maximum-length sub-sequence containing x_i should not be counted twice when it is not strictly larger than len. □

Changing one value in an instantiation modifies its focus cardinality of at most one.

Property 4. Let $I[X] = \langle v_0, v_1, \ldots, v_{n-1}\rangle$ be an instantiation of focus cardinality v_c and $x_i \in X$, and $I'[X] = \langle v'_0, v'_1, \ldots, v'_{n-1}\rangle$ be the instantiation such that $\forall j \in \{0, 1, \ldots, n-1\}, j \neq i$, $v_j = v'_j$ and: *(1)* If $v_i > k$ then $v'i \leq k$. *(2)* If $v_i \leq k$ then $v'_i > k$. The focus cardinality v'_c of $I'[X]$ is such that $|v_c - v'_c| \leq 1$.

Algorithm 3: MINCARDS($X = \langle x_0, x_1, \ldots, x_{n-1} \rangle, len, k$): Integer matrix

```
1  cards := new Integer[|X|][3] ;
2  if min(x_0) ≤ k ∧ max(x_0) > k then
3      cards[0][0] := 0;
4      cards[0][1] := 1;
5      cards[0][2] := 1;
6  else
7      if min(x_0) > k then
8          cards[0][0] := n + 1;
9          cards[0][1] := 1;
10         cards[0][2] := 1;
11     else
12         cards[0][0] := 0;
13         cards[0][1] := n + 1;
14         cards[0][2] := 0;
15 for Integer i := 1; i < n; i := i + 1 do
16     if max(x_i) > k then
17         if min(x_i) > k then cards[i][0] := n + 1;
18         else cards[i][0] := min(cards[i − 1][0], cards[i − 1][1]);
19         if cards[i − 1][2] = 0 ∨ cards[i − 1][2] = len then
20             cards[i][1] := min(cards[i − 1][0] + 1, cards[i − 1][1] + 1);
21             cards[i][2] := 1;
22         else
23             cards[i][1] := min(cards[i − 1][0] + 1, cards[i − 1][1]);
24             if cards[i − 1][1]<cards[i − 1][0]+1 then cards[i][2] := cards[i−1][2]+1;
25             else cards[i][2] := 1;
26     else
27         cards[i][0] := min(cards[i − 1][0], cards[i − 1][1]);
28         cards[i][1] := n + 1;
29         cards[i][2] := 0;
30 return cards;
```

Proof. Assume first that $v_i > k$. x_i belongs to a $\{\mathrm{P}_k\}$-sequence p. Let s be the length of this $\{\mathrm{P}_k\}$-sequence within $I[X]$. We can split p into $p_1 = \langle x_k, x_{k+1}, \ldots, x_{i-1} \rangle$, $p_2 = \langle x_i \rangle$, $p_3 = \langle x_{i+1}, x_{i+1}, \ldots, x_l \rangle$ ($p1$ and/or p_3 can be empty). Let q_1, q_3 and r_1, r_3 be positive or null integers such that $r_1 < len, r_3 < len$ and $s = q_1 \times len + r_1 + 1 + q_3 \times len + r_3$. By construction, the maximum contribution of the variables in p to the focus cardinality of $I'[X]$ (that is, with $x_i \leq k$), is equal to $q_1 + 1 + q_3 + 1 = q_1 + q_3 + 2$. With respect to $I[X]$, the contribution is then equal to $q_1 + q_3 + \lceil \frac{r_1 + r_3 + 1}{len} \rceil$. The minimum value of $\lceil \frac{r_1 + r_3 + 1}{len} \rceil$ is 1. In this case the property holds. The minimum contribution of the variables in p to the focus cardinality of $I'[X]$ is equal to $q_1 + q_3$. In this case, with respect to $I[X]$ the maximum value of $\lceil \frac{r_1 + r_3 + 1}{len} \rceil$ is $\lceil \frac{1}{len} \rceil = 1$, the property holds. The last intermediary case is when the contribution of the variables in p to the focus cardinality of $I'[X]$ is equal to $q_1 + q_3 + 1$. The minimum value of $\lceil \frac{r_1 + r_3 + 1}{len} \rceil$ is 1 and its maximum is 2, the property holds. The reasoning for $v_i \leq k$ is symmetrical. □

From Property 4, we know that domains of variables in X can be pruned only once y_c is fixed since the variation coming from a single variable in X is at most one.

Algorithm 4 shrinks $D(y_c)$ and all the variables in X in $O(n)$. It first calls Algorithm 3 to obtain $minCard(X)$ from $\min(\underline{p}(x_{n-1}, v_>)$ and $\underline{p}(x_{n-1}, v_\leq))$ and eventually shrinks $D(y_c)$. Then, it computes the data for suffixes, and uses Property 3 to reduce domains of variables in X according to $\max(y_c)$. Since removed values of variables in X cannot lead to a focus cardinality strictly less than $\max(y_c)$, it enforces GAC. Algorithm 4 does not directly modify domains: X and y_c are locally copied, and the filtered copies are returned. The reason is that we will use this algorithm in an extension of FOCUS in Section 5. To improve the readability, we assume that the solver raises an exception FAILEXCEPTION if a domain of one copy becomes empty.

Algorithm 4: FILTER($X = \langle x_0, x_1, \ldots, x_{n-1}\rangle, y_c, len, k$): Set of variables

```
1  cards := MINCARDS(X, len, k) ;
2  Integer lb := min(cards[n − 1][0], cards[n − 1][1]);
3  if min(y_c) < lb then D(y_c) := D(y_c) \ [min(y_c), lb[;
4  if min(y_c) = max(y_c) then
5      sdrac := MINCARDS(⟨x_{n−1}, x_{n−2}, ..., x_0⟩, len, k) ;
6      for Integer i := 0; i < n; i := i + 1 do
7          if cards[i][0] + sdrac[n − 1 − i][0] > max(y_c) then
8              D(x_i) := D(x_i)\ [min(x_i), k];
9          Integer regret := 0;
10         if cards[i][2] + sdrac[n − 1 − i][2] − 1 ≤ len then regret := 1;
11         if cards[i][1] + sdrac[n − 1 − i][1] − regret > max(y_c) then
12             D(x_i) := D(x_i)\ ]k, max(x_i)];
13 return X ∪ {y_c};
```

Example 3. Consider FOCUS($X = \langle x_0, x_1, \ldots, x_4\rangle, y_c, len, 0$) and $D(y_c) = \{1, 2\}$. *(1)* Assume $len = 2$ and $D(x_0)$=$D(x_2)$=$D(x_3)$=$\{1, 2\}$, $D(x_1) = \{0\}$ and $D(x_4)$ =$\{0, 1, 2\}$. Line 3 of Algorithm 4 removes $[1, 2[$ from $D(y_c)$. Since the length of the $\{\mathrm{P}_k\}$-sequence $\langle x_2, x_3\rangle$ is equal to *len*, *cards*[4][1]=3 and *cards*[4][2]=1. *sdrac* [5−1−4][1]=1 and *sdrac*[5−1−4][2]=1. *regret*=1 and thus 3+1−*regret*=3>max(y_c), $]0, 2]$ is removed from $D(x_4)$ (line 12). *(2)* Assume now *len*=3 and $D(x_0)$=$D(x_2)$ =$D(x_4)$=$\{1, 2\}$, $D(x_1)$=$\{0\}$ and $D(x_3)$=$\{0, 1, 2\}$. Line 3 of Algorithm 4 removes $[1, 2[$ from $D(y_c)$. Since value 0 for x_3 leads to a focus cardinality of 3 (*cards*[3][0] =2 and *sdrac*[5−1−3][0]=1), strictlty greater than $\max(y_c)$, $\langle x_2, x_3, x_4\rangle$ must be a $\{\mathrm{P}_k\}$-sequence (of length $3 \leq len$). Algorithm 4 removes value 0 from $D(x_3)$ (line 8). ⊛

5 Constraints Derived from FOCUS

5.1 Using a Variable for *len*

Assume that, in Example 1 of the Introduction, several companies offer leases with different maximum duration. We aim at computing the best possible configuration for each different offer, based on each maximum duration. To deal with this case, we can extend FOCUS so as to define *len* as a variable, with a discrete domain since the maximum durations of rentals are proper to each company. Another use of this extension is the

Algorithm 5: FILTERVARLEN($X = \langle x_0, x_1, \ldots, x_{n-1} \rangle$, y_c, len, k): Set of variables

```
1  IntegerVariable[][] vars := new IntegerVariable[|D(len)|][];
2  Integer j := 0;
3  foreach v_l ∈ D(len) do
4      try vars[j] := FILTER(X, y_c, v_l, k); // the last variable is y_c
5      catch FAILEXCEPTION: D(len) := D(len) \ {v_l}; // in this case vars[j] = null
6      j := j + 1;
7  Integer min_c := max(y_c) + 1;
8  j := 0;
9  foreach v_l ∈ D(len) do
10     if vars[j] ≠ null then min_c := min(min_c, min(vars[j][n]));
11     j := j + 1;
12 D(y_c) := D(y_c) \ [min(y_c), min_c[;
13 for Integer i := 0; i < n; i := i + 1 do
14     Integer min_i := max(x_i) + 1;
15     Integer max_i := min(x_i) − 1;
16     j := 0;
17     foreach v_l ∈ D(len) do
18         if vars[j] ≠ null then
19             min_i := min(min_i, min(vars[j][i]));
20             max_i := max(max_i, max(vars[j][i]));
21         j := j + 1;
22     D(x_i) := D(x_i) \ ([min(x_i), min_i[ ∪ ]max_i, max(x_i)]) ;
23 return X ∪ {y_c} ∪ {len};
```

case where the end-user wishes to compare for the same company several maximum package duration, enumerate several solutions, etc.

The filtering algorithm of this extension of FOCUS uses following principle: For each value v_l in $D(len)$, we call FILTER(X, y_c, v_l, k) (Algorithm 4). If an exception FAILEXCEPTION is raised, v_l is removed from $D(len)$. Otherwise, we store the result of the filtering. At the end of the process, value $v \in D(x_i)$, $x_i \in X$, is removed from its domain if and only if it was removed by all the calls to FILTER(X, y_c, v_l, k) that did not raised an exception. Algorithm 5 implements this principle. Since it calls Algorithm 4 for each value in $D(len)$, it enforces GAC. Its time complexity is $O(n \times |D(len)|)$.

5.2 Harder constraint on y_c

In Definition 1, the number of sequences in S could be constrained by an equality: $|S| = v_c$. When the maximum value for the variable y_c is taken, the number of counted disjoint sequences of length at most len is maximized. This maximum is equal to the number of $\{\mathrm{P}_k, \mathrm{U}_k\}$-sequences (we consider sequences of length one). The filtering is obvious. If, in addition, we modify the condition 3 of Definition 1 to make it stronger, for instance $3 \leq |s_i| \leq len$, it remains possible to compute recursively the maximum possible number of disjoint sequences by traversing X, similarly to the Lemmas used for the focus cardinality. Conversely, the aggregation of prefix and suffix data to obtain an algorithm in $O(n)$ is different. We did not investigate this point because we are not convinced of the practical significance of this variation of FOCUS.

6 Discussion: Related Work and Decomposition

Although it seems to be similar to a specialization of GROUP,[2] the FOCUS constraint cannot be represented using GROUP because of len: Using FOCUS, the variable in the sequence which directly precedes (or succeeds) a counted group of values strictly greater than the parameter k can also take itself a value strictly greater than k, which violates the notion of group.

To remain comparable with a filtering algorithm having a time complexity linear in the number of variables, the automaton-based reformulation of FOCUS should directly manipulate an automaton with counters, which are used to estimate the focus cardinality. We selected the paradigm presented in [1]. Under some conditions, this framework leads to a reformulation where a complete filtering is achieved, despite the counters.[3] This paradigm is based on automata derived from constraint checkers. We propose an automaton $\mathcal{A}$ representing FOCUS deduced from Algorithm 1, depicted by Figure 3.

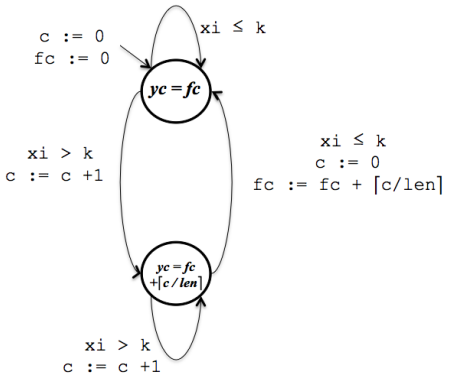

Fig. 3. Automaton with two counters c and fc, representing FOCUS. The two states are terminal. x_i denotes a variable in X and we consider the variable y_c and the value len of FOCUS.

As it is shown by figure 3, the automaton has two terminal states and maintains two counters c and fc, which represent respectively the size of the current traversed $\{\mathrm{P}_k\}$-sequence and the focus cardinality. Therefore, when $i = n - 1$, fc is compared to the value of y_c. The principle is the same than in Algorithm 1: Each time a $\{\mathrm{P}_k\}$-sequence ends, its contribution is added to the counter representing the focus cardinality. The automaton has two states and two counters. The constraint network $\mathcal{N}$ (hypergraph) encoding the automaton [1] is not Berge-Acyclic [4]. Propagating directly the constraints in $\mathcal{N}$ does not necessarily entail a complete filtering. However, within the automaton $\mathcal{A}$, the choice of the next transition only depends on the value of $x_i \in X$. Therefore, in $\mathcal{N}$, no signature constraints share a variable. From [1, p. 348-349], by enforcing pairwise consistency on the $O(n)$ pairs of transition constraints sharing more than one

[2] http://www.emn.fr/z-info/sdemasse/gccat/Cgroup.html

[3] Conversely to the COSTREGULAR constraint [3], for instance.

variable, we obtain a complete filtering. Three variables are shared: one representing the possible next states, and the variables for the two counters. In the worst case, pairwise consistency considers all the tuples for the shared variables. Counter variables (fc and c) have initially a domain of order $O(n)$ while the third domain is in $O(1)$, which leads to a time complexity in $O(n^3)$ for filtering FOCUS. When len and $\max(len)$ are small, this decomposition could be used. Thus, it is a contribution. The behavior with generic search strategies could be different for the decomposition and for the linear algorithm. However, using the decomposition requires to enforce pairwise consistency, which is not simpler and less generic than a dedicated filtering algorithm, because of events propagation. The decomposition is dependent on the priority rules of the solver.

7 Experiments

Since our filtering algorithm is complete, in $O(n)$, and without heavy data structures, performing benchmarks of the constraint isolated from a problem is not relevant. This section analyzes the impact of FOCUS on the solutions of the *sorting chords* problem described in Section 3. We selected this example rather than, for instance, the cumulative problem of Section 3, because all the other constraints (ALLDIFF as well as the ternary table constraints) can be provided with a complete filtering algorithm. With respect to the variable sum, we define it as the objective to minimize. Thus, instead of an equality we enforce the constraint $sum \leq \sum cost_i$, which is also tractable. We use the solver Choco (http://www.emn.fr/z-info/choco-solver/) with the default variable and value search strategies (DomOverWDeg and MinValue), on a Mac OSX 2.2 GHz Intel Core i7 with 8GB of RAM memory. We are interested in two aspects:

1. Our constraint is used for refining an objective criterion and, from Property 4, we know that domains of variables in X can be pruned only once y_c is fixed. An important question is the following: Is our filtering algorithm significantly useful during the search, compared with a simple checker?
2. Are the instances harder and/or the value of the objective variable sum widely increased when we restrict the set of solutions by imposing FOCUS?

7.1 Pruning Efficacy

In a first experiment, we run sets of 100 random instances of the *sorting chords* problem, in order to compare the complete filtering of FOCUS with a simple checker. For each set, the maximum value of y_c is either 1 or 2, in order to consider instances where FOCUS as a important influence. We compare the pruning efficacy of FOCUS with a light version of FOCUS, where the propagation is reduced to the checker.

Table 1 summarizes the results on 8 and 9 chords (respectively 17 and 19 variables in the problem). The instances are all satisfiable and optima were always proved. With larger instances ($\geq$ 10 chords), optimum cannot be proved in a reasonable number of backtracks using the checker. In the table, $\overline{y_c} = \max(y_c)$, and len and k are the parameters of FOCUS. $nmax$ is the maximum possible number of changing notes between any two chords. Average values are given as integers. Table 1 clearly shows that,

even with a GAC filtering algorithm for ALLDIFF and for the ternary table constraints COSTC$(ch_i, ch_{i+1}, cost_i)$, we can only solve small instances without propagating FOCUS, some of them requiring more than one million backtracks. Conversely, the number of backtracks for proving optimality is small and stable when Algorithm 4 is used for propagating FOCUS. Using the filtering algorithm of FOCUS is mandatory.

Instances			FOCUS(X, y_c, len, k)				CHECKER(X, y_c, len, k)			
nb. of chords	$\overline{y_c}$-len -k -$nmax$	#optimum with $sum > 0$	average #backtracks (of 100)	average #fails (of 100)	max. #backtracks (of 100)	average time (ms) (of 100)	average #backtracks (of 100)	average #fails (of 100)	max. #backtracks (of 100)	average time (ms) (of 100)
8	1-4-0-3	66	61	46	462	11	1518	951	15300	17
8	1-4-1-3	66	45	33	342	1	91	59	1724	2
8	2-4-0-3	66	47	34	247	1	58	42	455	1
8	2-4-1-3	66	45	33	301	1	44	32	349	1
8	1-6-0-4	84	198	141	2205	12	15952	10040	86778	213
8	1-6-1-4	84	114	79	767	3	1819	1117	45171	24
8	2-6-0-4	84	127	88	620	3	2069	1361	64509	28
8	2-6-1-4	84	118	81	724	3	250	168	7027	4
8	1-8-0-5	98	261	184	1533	6	39307	25787	167575	566
8	1-8-1-5	98	148	103	662	3	11821	7642	94738	168
8	2-8-0-5	98	164	113	803	4	21739	14400	173063	317
8	1-8-0-5	98	183	127	882	4	10779	6939	92560	153
8	1-8-0-6	99	290	203	1187	18	46564	30488	130058	690
8	1-8-1-6	99	238	166	1167	11	29256	19150	134882	438
8	2-8-0-6	99	221	152	1458	6	29455	19607	123857	445
8	2-8-1-6	99	209	144	1118	9	21332	14095	117768	329
9	1-9-0-4	88	415	299	4003	18	214341	133051	1095734	3244
9	1-9-1-4	88	268	185	2184	6	12731	7988	751414	203
9	2-9-0-4	88	270	188	2714	6	22107	14065	374121	337
9	2-9-1-4	88	266	182	3499	6	1364	941	92773	23
9	1-9-0-5	97	574	407	2437	26	360324	230167	1355934	6584
9	1-9-1-5	97	404	273	1677	11	62956	40277	881441	1150
9	2-9-0-5	97	451	309	3327	12	228072	147007	1124630	4263
9	2-9-1-5	97	386	260	1698	10	58421	37589	989900	1079

Table 1. Comparison of Algorithm 4 with a checker, on the *sorting chords* problem. Each row represents 100 randomly generated instances. $\overline{y_c}$ is $\max(y_c)$. $nmax$ indicates the maximum possible common notes between two chords. Optimum solutions were found for all the considered instances. "#backtracks" means the number of backtracks, "#fails" the number of fails, "#optimum with $sum > 0$" is the number of solutions with a non null objective value.

7.2 Impact on the Objective Value

In a second experiment, we run sets of 100 random instances of the *sorting chords* problem, in order to compare problems involving FOCUS with the same problems where FOCUS is removed from the model. The goal of this experiment is to determine, with respect to the sorting chords problem, whether the optimum objective value increases widely or not when FOCUS is imposed, as well as the time and backtracks required to obtain such an optimum value. Table 2 summarizes the results with a number of chords varying from 6 to 20 (13 to 51 variables in the problem). The instances are all satisfiable and optima were always found and proved. In each set, we count the number of instances with distinct objective values. For sake of space, we present the results for $k = 0$ and $len = 4$. Similar results were found for closed values of k and len. Despite the instances with FOCUS are more constrained with respect to the variables involved in the objective, Table 2 does not reveals any significant difference, both with respect to the number of backtracks (and solving time) and the optimum objective value. We finally have differences in objective values at most equal to 2.

To complete our evaluation, we search for the first solution of larger problems, in order to compare the scale of size that can be reached with and without using FOCUS.

Instances				with FOCUS				without FOCUS			
nb. of chords	$\overline{y_c}$ -len -k -$nmax$	#optimum with $sum > 0$	#optimum equal with / without FOCUS	max. value of sum	average #backtracks (of 100)	max. #backtracks (of 100)	average time (ms) (of 100)	max. value of sum	average #backtracks (of 100)	max. #backtracks (of 100)	average time (ms) (of 100)
6	2-4-0-4	88	99	7	23	76	3	7	22	76	1
8	2-4-0-4	84	95	8	131	618	4	7	115	449	3
10	2-4-0-4	78	98	6	457	4579	11	6	360	3376	9
12	2-4-0-4	69	98	5	952	12277	28	5	1010	10812	27
16	2-4-0-4	43	100	4	4778	132019	153	4	6069	95531	189
20	2-4-0-4	7	100	3	15650	1316296	747	3	15970	1095399	679
6	2-4-0-6	97	96	13	37	113	1	13	37	121	1
8	2-4-0-6	99	93	11	218	1305	5	11	198	860	4
10	2-4-0-6	97	77	10	1247	5775	32	9	1159	10921	26
12	2-4-0-6	96	75	12	5092	34098	155	11	4844	54155	145
16	2-4-0-6	88	84	9	45935	724815	2002	9	73251	2517570	3407
20	2-4-0-6	79	91	8	264881	4157997	14236	6	174956	2918335	8284

Table 2. Comparison of instances of the *sorting chords* problem with and without FOCUS. The column "#optimum equal with and without FOCUS" indicates the number of instances for which, with and without FOCUS, the optimum objective value is equal. "max. value of sum" indicates the maximum objective value among the 100 instances.

The value heuristic assigns first the smaller value, which is semantically suited to the goal of the problem although we do not search for optimum solutions.

Instances			with FOCUS				without FOCUS			
nb. of chords	$\overline{y_c}$ -len -k -$nmax$	average gap in sum	min(sum) / max(sum) (of 100)	average #backtracks (of 100)	max. #backtracks (of 100)	average time (ms) (of 100)	min(sum) / max(sum) (of 100)	average #backtracks (of 100)	max. #backtracks (of 100)	average time (ms) (of 100)
50	4-6-2-4	14	45/73	0	16	84	55/94	0	0	80
100	8-6-2-4	27	100/145	1	57	1168	119/175	0	0	1413
50	5-6-1-4	30	19/69	212	12698	101	55/94	0	0	85
100	10-6-1-4	63	40/113	93	3829	1002	119/175	0	0	1407

Table 3. Comparison of the results for providing a first solution on large instances of the *sorting chords* problem, with and without FOCUS. The "average gap in sum" is the average of the difference, for each of the 100 instances, between the objective value without FOCUS and the objective value with FOCUS. This latter one (with FOCUS) is smaller or equal for all the instances.

Results are compared in Table 3. They show that, with FOCUS, the number of backtracks grows for some instances when the parameters of FOCUS are shrunk (one of the instance with 50 chords required 12698 backtracks). However, the objective value of the first solution is systematically significantly better when FOCUS is set in the model. One explanation of these results is the following: Focusing the costs on a small number of areas within the sequence semantically tends to limit the value of their total sum.

8 Conclusion

We presented FOCUS, a new constraint for concentrating high costs within a sequence of variables. In the context of scheduling, many use cases for the constraint make sense. We proposed a $O(n)$ complete filtering algorithm, where n is the number of variables. We proposed an automaton-based decomposition and we discussed extensions of our constraint. Our experiments investigated the importance of propagating FOCUS and the impact of FOCUS on the solutions of problems. Our results demonstrated that the complete filtering algorithm of FOCUS is mandatory for solving instances.

Acknowledgments We thank Jean-Guillaume Fages for the helpful comments he provided on the paper.

References

1. N. Beldiceanu, M. Carlsson, R. Debruyne, and T. Petit. Reformulation of global constraints based on constraints checkers. *Constraints*, 10(4):339–362, 2005.
2. A. De Clercq, T. Petit, N. Beldiceanu, and N. Jussien. Filtering algorithms for discrete cumulative problems with overloads of resource. *Proc. CP*, pages 240–255, 2011.
3. S. Demassey, G. Pesant, and L.-M. Rousseau. A cost-regular based hybrid column generation approach. *Constraints*, 11(4):315–333, 2006.
4. P. Janssen and M-C. Vilarem. Problèmes de satisfaction de contraintes: techniques de résolution et application à la synthèse de peptides. *Research Report C.R.I.M.*, 54, 1988.
5. F. Pachet and P. Roy. Musical harmonization with constraints: A survey. *Constraints*, 6(1):7–19, 2001.
6. G. Pesant and J.-C. Régin. Spread: A balancing constraint based on statistics. *Proc. CP*, pages 460–474, 2005.
7. T. Petit and E. Poder. Global propagation of side constraints for solving over-constrained problems. *Annals of 0perations Research*, pages 295–314, 2011.
8. T. Petit and J.-C. Régin. The ordered distribute constraint. *International Journal on Artificial Intelligence Tools*, 20(4):617–637, 2011.
9. T. Petit, J-C. Régin, and C. Bessière. Meta constraints on violations for over constrained problems. *Proc. IEEE-ICTAI*, pages 358–365, 2000.
10. J-C. Régin. A filtering algorithm for constraints of difference in CSPs. *Proc. AAAI*, pages 362–367, 1994.
11. P. Schaus, Y. Deville, P. Dupont, and J-C. Régin. The deviation constraint. *Proc. CPAIOR*, 4510:260–274, 2007.
12. P. Schaus, P. Van Hentenryck, and J-C. Régin. Scalable load balancing in nurse to patient assignment problems. *Proc. CPAIOR*, 5547:248–262, 2009.
13. C. Truchet and P. Codognet. Musical constraint satisfaction problems solved with adaptive search. *Soft Comput.*, 8(9):633–640, 2004.